BĂNG TẢI GIAO HÀNH LÝ THÔNG MINH SỬ DỤNG KẾT HỢP PHÁT HIỆN VÀ NHẬN DẠNG MÃ VẠCH

SMART LUGGAGE DELIVERY CONVEYER USING BARCODE DETECTION AND DECODE

Đoàn Thị Hương Giang[1,*]

DOI: https://doi.org/10.57001/huih5804.2023.076

TÓM TẮT

Ứng dụng kết hợp thành tựu của trí tuệ nhân tạo vào trong hệ thống tự động hóa ngày càng được các nhà phát triển sản phẩm quan tâm. Trong đó, việc giao hành lý cho hành khách sau một chuyến bay đảm bảo sự an toàn, an ninh, chính xác và thuận tiện là một trong những yêu cầu cấp bách. Tuy nhiên, hệ thống giao hành lý ở các sân bay hiện nay rất dễ xảy ra hiện tượng lấy nhầm hoặc thất lạc hành lý khi cùng một chuyến bay có hai hành lý có hình thức giống nhau. Hoặc một lý dó khác là các hành khách phải đứng xung quanh các băng tải để tìm hành lý của mình mất khá nhiều công sức, thời gian, gây cảm giác mệt mỏi, khó chịu cho khách hàng. Cùng với sự phát triển như vũ bão của kỹ thuật điện tử, mạng internet và trí tuệ nhân tạo thì việc thông minh hóa quá trình giao hành lý trong sân bay có thể thực hiện thông qua thông tin hình ảnh và trí tuệ nhân tạo. Bài báo này tác giả đề xuất một hệ thống điều khiển hoàn thiện cho băng tải giao hành lý sân bay thông minh. Hệ thống thông minh đề xuất kết hợp tự động hóa, điện-điện tử và nhận dạng mã vạch tự động có thời gian đáp ứng nhanh, bền vững và độ chính xác cao.

Từ khóa: *Băng tải tự động, học sâu, phát hiện mã vạch, giải mã mã vạch, tương tác người - máy, mạng nơ ron tích chập.*

ABSTRACT

Combining of artificial intelligence into the automation system is increasingly interested of the technology companies. In particular, the delivery of luggage to passengers after a flight to ensure safety, security, accuracy and convenience is one of the urgent requirements. However, the current system of baggage delivery system at the airports is usually wrong or lost luggage which two luggages are the same form, size, and style. Another problem, the passengers must wait and stand around the conveyors to find their luggages, which takes a lot of effort and time, causing fatigue and discomfort for customers.. With the rapid development of electronic technology, the internet, and artificial intelligence, the smartening of the baggage delivery process in the airport need deploy. In this research, we will propose a complete conveyor control system in order to deliver intelligent baggage delivery in the airport. Our propose system combines of automation, electro-electronics and barcode recognition that reachs the realtime time, high accuracy and robust system.

Keywords**:** *Automatic Conveyer, deep learning, barcode detection, decode, human-machine interaction, convolution Neuron network.*

[1]Khoa Điều khiển và Tự động hóa, Đại học Điện lực

[*]Email: giangdth@epu.edu.vn

Ngày nhận bài: 18/12/2022

Ngày nhận bài sửa sau phản biện: 30/3/2023

Ngày chấp nhận đăng: 26/4/2023

1. GIỚI THIỆU

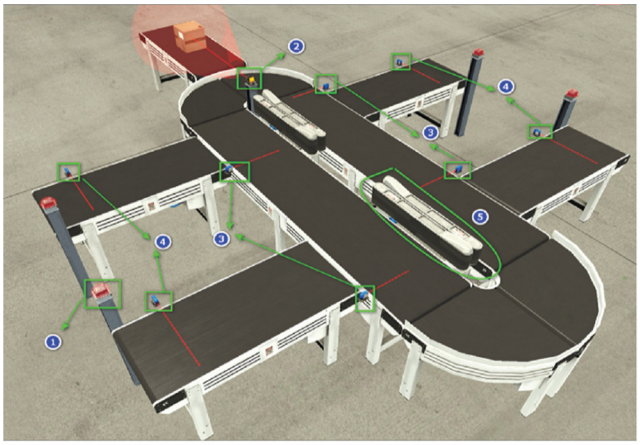
Cùng với sự phát triển của kỹ thuật điện - điện tử, tự động hóa thì băng chuyền vận chuyển trả hành lý tự động đã được sử dụng trong các sân bay để đưa hành lý tới hành khách được triển khai từ rất lâu trên tất cả các sân bay. Cho đến nay, các hệ thống băng chuyền tự động vẫn đang được áp dụng và ngày càng đước các hãng công nghệ cải tiến [17]. Các mẫu băng chuyền phổ biến nhất được các sân bay sử dụng gồm băng chuyền hình ô van, hình tròn, nhiều tầng,... Hành lý được đưa ra băng chuyền từ kho tiếp vận. Sau đó, các hành khách có thể đứng quanh khu vực quanh băng chuyền để tìm và tự lấy hành lý của mình. Hành lý sẽ được quay vòng liên tục nếu chúng chưa được đưa ra khỏi băng chuyền. Với hệ thống băng chuyền như vậy tồn tại nhiều vấn đề phát sinh như: (1) thất lạc hành lý do người khác lấy nhầm các hành lý có mẫu mã giống nhau, (2) sau chuyến đi dài hành khách vẫn phải đứng và tìm hành lý trong số rất nhiều các hành lý khác nhau, gây mệt mỏi cho khách hàng. Trong bài báo này, chúng tôi đưa ra một giải pháp giao hành lý thông minh sử dụng kết hợp nhận dạng mã vạch (barcode) và thẻ từ (RFID)[14, 19]. Hệ thống này gồm băng tải giao hành lý tự động, hệ thống tự động phát hiện và giải mã mã vạch tự động dung trí tuệ nhân tạo và xử lý ảnh, và hệ thống đọc thẻ từ tự động. Hệ thống đề xuất nhằm thực hiện một cách tự động, chính xác và đem tới sự thuận tiện cho khách hàng. Tuy nhiên, mô hình phát hiện và giải mã mã vạch cần phải có độ chính chính xác cao và đáp ứng thời gian nhanh. Ngoài ra, hệ thống cần có sự kết nối giữa các khâu một cách tổng thể, đồng bộ.

Như chúng ta đã biết, mạng nơ ron tích chập (Convolution Neural Networks) ngày càng tập trung sự quan tâm của nhiều nhà khoa học do những kết quả vượt trội của chúng so với các giải pháp học máy truyền thống (Machine Learning). Đặc biệt trong lĩnh vực phát hiện và nhận dạng đối tượng trong ảnh thì mạng YOLO đem đến những bước tiến vượt trội so với những giải pháp truyền thống như Dlib [15], FastRCNN [16]. Do đó, trong nghiên cứu này chúng tôi đề xuất dùng mạng YOLO[3-8] và Pyzbar[13] để khảo sát và đưa ra phương án lựa chọn phù hợp với giải pháp tổng thể của hệ thống giao hành lý sân bay thông minh. Trong nghiên cứu này chúng tôi cũng sẽ tiến hành thu thập một bộ cơ sở dữ liệu (CSDL) mã vạch,

sau đó gán nhãn theo chuẩn phù hợp để huấn luyện các định dạng của mạng YOLO. CSDL đã gán nhãn này được công bố cho cộng đồng nghiên cứu để có thể sử dụng miễn phí. Mô hình hệ thống sau đó sẽ được triển khai và đánh giá về độ chính xác và thời gian đáp ứng một cách tổng thể.

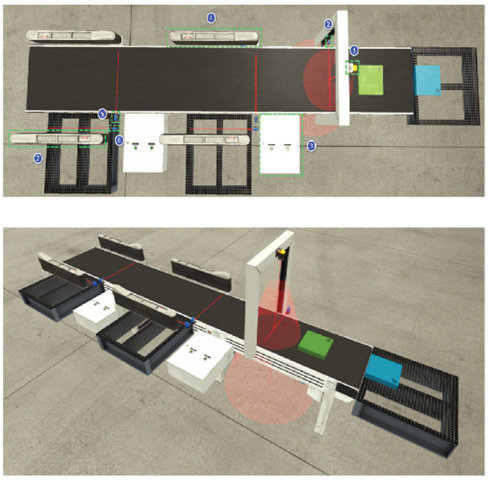
Phần tiếp theo của bài báo gồm các phần sau đây: Phần 2 mô tả chi tiết giải pháp đề xuất. Kết quả thử nghiệm được trình bày trong phần 3. Phần 4 là mục cuối cùng sẽ trình bày kết luận và hướng phát triển trong thời gian tiếp theo

2. GIẢI PHÁP ĐỀ XUẤT



- (1) – Camera
- (2) – Cảm biến phát hiện hành lý tại điểm chụp ảnh
- (3) – Bốt quẹt thẻ
- (4) – Tay gạt sử dụng động cơ sevor đưa hành lý vào điểm nhận hành lý
- (5) – Cảm biến phát hiện hành lý đã đến bot quẹt thẻ
- (6) – Cảm biến phát hiện hành lý đã ra hàng chờ
- (7) – Tay gạt sử dụng động cơ servo đưa hành lý trở lại băng tải thẳng

a) Ý tưởng



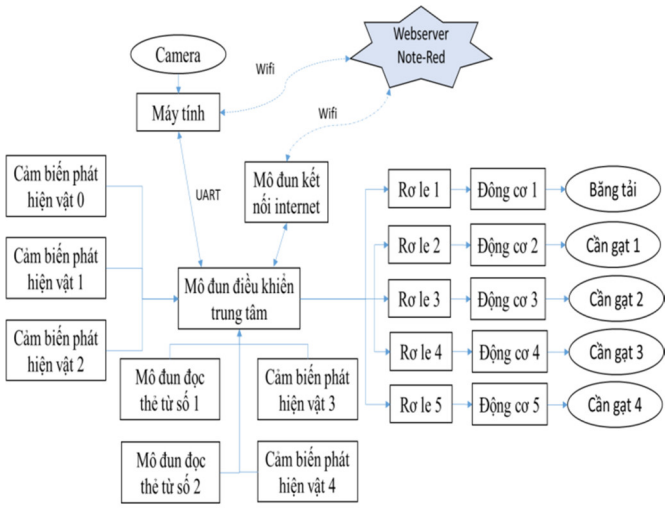
- (1) – Camera
- (2) – Cảm biến phát hiện hành lý tại điểm chụp ảnh
- (3) – Bốt quẹt thẻ
- (4) – Tay gạt sử dụng động cơ sevor đưa hành lý vào điểm nhận hành lý
- (5) – Cảm biến phát hiện hành lý đã đến bot quẹt thẻ
- (6) – Cảm biến phát hiện hành lý đã ra hàng chờ
- (7) – Tay gạt sử dụng động cơ servo đưa hành lý trở lại băng tải thẳng

b) Mô hình

Hình 1. Sơ đồ hệ thống giao hành lý sân bay thông minh

Ý tưởng hệ thống xây dựng cho mỗi hành lý sẽ được gắn tự động một mã vạch và hành khách sẽ được phát một thẻ RFID tương ứng với vé máy bay và tương ứng với một mã vạch. Hệ thống mô hình đề xuất của băng tải thông minh như minh họa trong hình 1(a). Tuy nhiên, khi triển khai mô hình băng tải tròn khá tốn kém nên mô hình thử nghiệm chúng tôi thiết kế là băng tải dài như hình 1(b) gồm các phần như sau: Xylanh đưa hành lý ra khỏi băng tải và đưa hành lý trở lại băng tải; Bốt quẹt thẻ RFID; Camera; Cảm biến quang phát hiện vật thể; Động cơ điều khiển băng tải; Băng tải; Máng chứa.

Khi hành khách quẹt thẻ tại một cột quẹt thẻ thì hệ thống sẽ kiểm tra xem có hành lý ở trên băng tải hay không nhờ vào các camera tại các điểm quét thẻ. Sau một khoảng thời gian mà hành lý của người quẹt thẻ không có trên hệ thống băng tải thì đèn chuyển trạng thái báo hành lý chưa đến. Nếu hệ thống nhận diện thấy có hành lý thì khi hành lý gần tới điểm quẹt thẻ thì đèn báo ở bốt đó sẽ sáng, và khi hành lý tới nơi, động cơ sẽ đẩy hành lý xuống máng chứa. Nếu sau một khoảng thời gian mà hành lý không được lấy khỏi máng chứa thì tay gạt sẽ gạt hành lý trở về băng tải. Sơ đồ khối hệ thống điều khiển được thể hiện như trong hình 2.



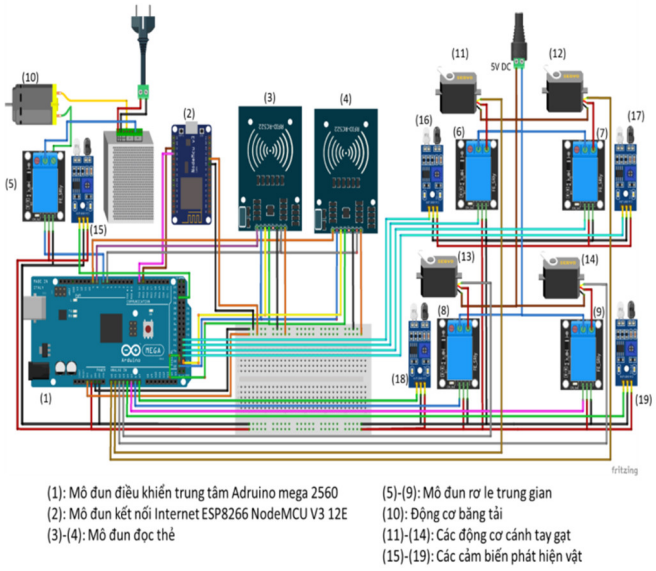
Hình 2. Sơ đồ khối mạch điều khiển mô hình băng tải giao hành lý thông minh trong sân bay

Mô hình hệ thống băng tải giao hành lý sân bay thông minh của chúng tôi đề xuất cho hai bốt giao hàng gồm các phần chính: (1) Thiết kế phần cứng mạch điều khiển băng tải tự động; (2) Phần mềm phát hiện và nhận dạng mã vạch; (3) Tích hợp mô hình hệ thống. Các nội dung này sẽ được trình bày chi tiết trong các mục tiếp theo của bài báo.

2.1. Thiết kế phần cứng mạch điều khiển

Cấu trúc phần cứng của hệ thống điều khiển băng tải thông minh được mô tả như trong sơ đồ khối ở hình 2. Trong đó, bộ điều khiển trung tâm là mô đun Adruino mega 2560[10] để đọc dữ liệu cảm biến gắn trên băng tải, nhận dữ liệu từ máy tính thông qua cổng UART và đồng thời mô đun điều khiển trung tâm cũng kết nối với mô đun ESP8266 NodeMCU V3 12E [19] để truyền dữ liệu quẹt thẻ

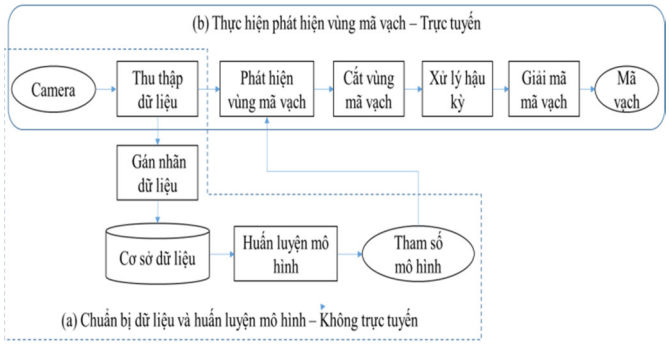
một cách tự động lên Webserver Note-Red giúp cung cấp thông tin cho quá trình quản lý dữ liệu hành lý. Đây là một công cụ cho phép kết nối thiết bị phần cứng, API và các dịch vụ Web với nhau. Việc truyền dữ liệu tự động từ mô đun điều khiển trung tâm lên Webserver thông qua địa chỉ đã được cài đặt đồng bộ ở cả mô đun ESP8266 NodeMCU V3 12E và Webserver. Mạch Arduino mega 2560 sẽ có tác dụng điều khiển ra các cơ cấu chấp hành khi kết hợp thông tin từ cảm biến, thẻ từ và thông tin hình ảnh từ máy tính. Có năm động cơ sử dụng trong mô hình gồm: 01 động cơ điều khiển băng tải là loại động cơ giảm tốc JGB37 – 555 60 rpm; 04 động cơ servo loại RC MG90S để điều khiển cánh tay gạt hành lý. Cảm biến phát hiện vật thể là loại cảm biến quang E18-D80NK. Sơ đồ đi dây và đấu nối phần cứng chi tiết của mạch điều khiển được biểu diễn như trong hình 3.



Hình 3. Cấu trúc ghép nối hệ thống

2.2. Phần mềm phát hiện và nhận dạng mã vạch

Tín hiệu từ camera chụp mã vạch được đưa về máy tính để phát hiện vùng mã vạch và sau đó sẽ bóc tách và giải mã để có mã vạch tương ứng. Quá trình này được thể hiện chi tiết như hình 4.



Hình 4. Sơ đồ khối hệ thống phát hiện và giải mã mã vạch

Ảnh chụp mã vạch thường chứa nền nên sẽ cho qua mạng YOLO để phát hiện vùng chứa mã vạch. Sau đó, vùng chứa mã vạch sẽ được cắt loại bỏ nền và xử lý hậu kỳ trước khi giải mã để có dãy số tương ứng đã được mã hóa theo tiêu chuẩn mã hóa. Tuy nhiên, đây là sơ đồ khối hệ thống nhận dạng mã vạch trực tuyến. Trước đó, hệ thống cần thực hiện các bước gồm: (1) chuẩn bị cơ sở dữ liệu mã vạch và gán nhãn để huấn luyện các hệ thống YOLO; (2) huấn luyện mô hình mạng YOLO; (3) giải mã mã vạch. Ba bước này được mô tả chi tiết như trong các mục sau đây.

2.2.1. Cơ sở dữ liệu mã vạch

Tác giả thực hiện thu thập CSDL mã vạch 9000 hình ảnh mã vạch tại các cửa hàng bán lẻ, siêu thị và nguồn Internet. Tập dữ liệu được gán nhãn định dạng chuẩn theo YOLO thông qua phần mềm gán nhãn LabelImg [11]. Công cụ này được viết bằng ngôn ngữ Python và sử dụng giao diện người dùng Qt giúp lưu nhãn đối tượng dưới dạng tệp XML ở định dạng PASCALVOC [20]. Bên cạnh đó, phần mềm LabelImg cũng hỗ trợ tạo CSDL phù hợp với các định dạng YOLO và CreateML. Trong nghiên cứu này, tác giả sử dụng định dạng phù hợp với YOLO để gán nhãn cho tập dữ liệu mã vạch do tự thu thập. Định dạng YOLO của dữ liệu mã vạch mà tác giả gán nhãn chứa tệp văn bản tương ứng với mỗi hình ảnh của mỗi nhãn mã vạch như minh họa trong hình 5.

Hình 5. Gán nhãn mã vạch bằng công cụ LabelImg

Cấu trúc mỗi nhãn mã vạch tác giả thực hiện gồm: <object-class> <x> <y> <w> <h>. Trong đó, <object-class> - lớp đối tượng là số thứ tự lớp đối tượng trong tập CSDL, bắt đầu từ 0. Trong bài báo này, tác giả thực hiện gán một nhãn cho mã vạch Barcode và các thử nghiệm được tiến hành với một nhãn là 0. Các giá trị <x> <y>; <w> <h> lần lượt là tọa độ tâm, chiều rộng, cao của vùng bao quanh mã vạch. Bộ CSDL mã vạch gán nhãn được đặt tên là BarcodeEpu và cung cấp cho cộng đồng nghiên cứu tại địa chỉ [1]. Cơ sở dữ liệu này được tác giả sử dụng để đánh giá hiệu quả của các mô hình YOLO [3-8]. Kết quả được phân tích, đánh giá làm cơ sở để lựa chọn mô hình phù hợp sẽ sử dụng trong toàn hệ thống điều khiển giao hành lý thông minh tại sân bay. Các đánh giá thử nghiệm sẽ được trình bày chi tiết trong phần 3.

2.2.2. Phát hiện và giải mã mã vạch

Phát hiện mã vạch: Như chúng ta đã biết, mạng YOLO là một trong những mạng nơ ron rất mạnh trong việc phát hiện và nhận dạng đối tượng trong ảnh. Đã có rất nhiều phiên bản YOLO khác nhau từ YOLOV1 [2] lần đầu xuất hiện năm 2015 cho đến bản hiện tại là YOLOV7 [8] công bố

năm 2022. Mỗi phiên bản có ưu điểm và nhược điểm riêng. Trong khuôn khổ bài báo này, tác giả không trình bày lại về các mạng YOLO mà sẽ sử dụng một số phiên bản từ YOLOV2 [3] đến YOLOV7 [8] để huấn luyện lại và đánh giá thử nghiệm cho phát hiện mã vạch. Tuy nhiên, mạng YOLO gốc được công bố cho cộng đồng dùng chung lại được huấn luyện trên bộ CSDL ImageNet [9], PASCAL VOC [20], và MS COCO [18] mà trong đó không phải huấn luyện cho đối tượng mã vạch. Do đó, trong bài báo này, tác giả sẽ sử dụng CSDL BarcodeEpu[1] tự thu thập và gán nhãn như trình bày trong mục 2.2.1 để huấn luyện lại mô hình và đánh giá thử nghiệm độ chính xác của quá trình phát hiện mã vạch. Kết quả thử nghiệm được trình bày trong mục 3.1. Toàn bộ quá trình từ thu thập vào gán nhãn CSDL đến huấn luyện mô hình được thực hiện không trực tuyến như khối có đường bao nét đứt của hình 4(a).

Hệ thống mạng YOLO sau khi đã huấn luyện (hình 4(a)) sẽ được sử dụng để chạy ở pha trực tuyến như minh họa trong khối đường bao nét liền của hình 4(b). Trong đó, đầu vào hệ thống trực tuyến là dữ liệu được thu thập trực tiếp từ camera của mô hình (ảnh thu nhận trực tiếp như hình 6(a)). Sau đó, hệ thống YOLO sẽ lấy dữ liệu đầu vào này để phát hiện vùng mã vạch (hình 6(b)) và cắt vùng mã vạch để loại bỏ bớt nền (hình 6(c)). Tuy nhiên, dữ liệu mã vạch thực tế thu nhận từ camera chụp trực tiếp thường không có chiều thẳng đứng như chúng ta mong muốn mà nó thường bị xoay như minh họa trong hình 6(c). Do vậy, tác giả sử dụng phương pháp nắn ảnh (warping image) và kết quả thu được như hình 6(d). Dữ liệu ảnh đã loại bỏ nền và xử lý hậu kỳ được thực hiện giải mã mã vạch ở bước sau.

Để đánh giá hiệu quả của mô hình YOLO, tác giả sử dụng các chỉ tiêu độ chính xác P (Precision), độ triệu hồi R (Recall) và Độ chính xác trung bình mAP (Mean Average Precision). Trong đó, độ chính xác thu được trên tập dữ liệu thử nghiệm là thước đo đánh giá độ hiệu quả của mô hình qua các thông số độ chính xác P và độ triệu hồi R. Độ chính xác P được định nghĩa là tỉ lệ của số mã vạch phát hiện đúng trên tổng số đối tượng được phát hiện minh họa bởi công thức sau đây:

$$P = \frac{TP}{TP + FP}$$

Độ triệu hồi R được định nghĩa là tỉ lệ của số mã vạch phát hiện đúng trên tổng số mã vạch có trong tập hình ảnh như minh họa bởi công thức sau đây:

$$R = \frac{TP}{TP + FN}$$

Độ chính xác trung bình mAP của N lớp và được định nghĩa theo công thức sau đây:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

Trong đó: TP (True positive) là tổng số dự đoán đúng được phân lớp vào đúng (positive); FP (False positive) tổng số dự đoán sai được phân lớp vào đúng (positive); FN (False negative) tổng số dự đoán sai được phân lớp vào sai (negative); và TN (True negative) là tổng số dự đoán đúng được phân lớp vào sai (negative), N là tổng số lớp.

Giải mã mã vạch: Trong nghiên cứu này, tác giả sử dụng thư viện PyzBar [13] để giải mã mã vạch. Hình 6 minh họa kết quả của quá trình phát hiện, xử lý dữ liệu và giải mã mã vạch với YOLO và Pyzbar.

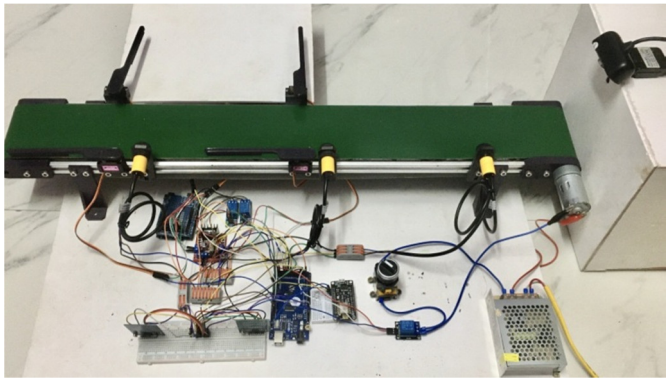
Hình 6. Kết quả minh họa quá trình phát hiện và giải mã mã vạch

Pyzbar là một thư viện mã nguồn mở được viết bằng ngôn ngữ Python, sử dụng phổ biến cho việc đọc mã vạch từ nhiều thiết bị khác nhau. Thư viện này hỗ trợ giải mã gần như toàn bộ các loại mã vạch như: EAN - 13, EAN - 8, UPC - A, UPC - E, Code 128, Code 39,… Kết quả đầu ra của quá trình giải mã là định dạng và dữ liệu mã vạch. Thư viện Pyzbar đã được các nhà nghiên cứu công bố với mô hình phù hợp cho bước giải mã mã vạch trong bài báo này. Đầu vào của thư viện Pyzbar là ảnh đầu ra của YOLO sau khi đã được xử lý hậu kỳ, đầu ra là dữ liệu và định dạng mã vạch. Ở phần xử lý hậu kỳ, dữ liệu sau khi cắt ra từ YOLO có thể bị xoay các góc khác nhau vì hành lý được sắp xếp ngẫu nhiên trong băng tải. Do các vùng chứa mã vạch luôn có dãy mã đặc biệt với những đặc điểm là những đường thẳng song song sát nhau trong diện tích hình chữ nhật. Tác giả sử dụng các phép biến đổi ảnh như lấy ngưỡng, làm dày đường biên để tạo vùng liên thông lớn nhất. Do các mã vạch là dãy các đường thẳng nằm sát kề nhau nên sẽ dễ dàng bị lấp đầy khi làm dày biên. Thuật toán tìm đường biên sau đó được áp dụng để phát hiện đường bao quanh vùng chứa mã vạch chính là vùng liên thông lớn nhất. Từ đó dễ dàng vẽ được hình chữ nhật bao quanh vùng liên thông lớn nhất này. Sau khi đã xác định cạnh của hình chữ nhật chứa mã vạch thì dễ dàng xác định hai điểm tương ứng với hai cạnh của chiều dài hình chữ nhật chứa vùng mã vạch. Như chúng ta đã biết, vùng mã vạch là một dãy các vạch nằm thẳng và vùng bao quanh mã vạch thường thuộc hình chữ nhật nằm ngang như hình 6 (d). Sau khi có được hai tọa độ ảnh của các đầu mút trên chiều dài hình chữ nhật bao quanh vùng mã vạch, tác giả tính toán được góc nghiêng của vùng mã vạch và tính ma trận xoay ảnh thông qua góc nghiêng nói trên. Cuối cùng, thuật toán warpAffine được áp dụng trên ảnh với ma trận xoay vừa tính được. Việc nắn và chỉnh góc nghiêng của ảnh được thực hiện hoàn toàn tự động trong phần mềm thông qua các thư viện của OpenCV. Trong nghiên cứu này, tác giả không tiến hành đánh giá chi tiết hiệu quả của riêng thư

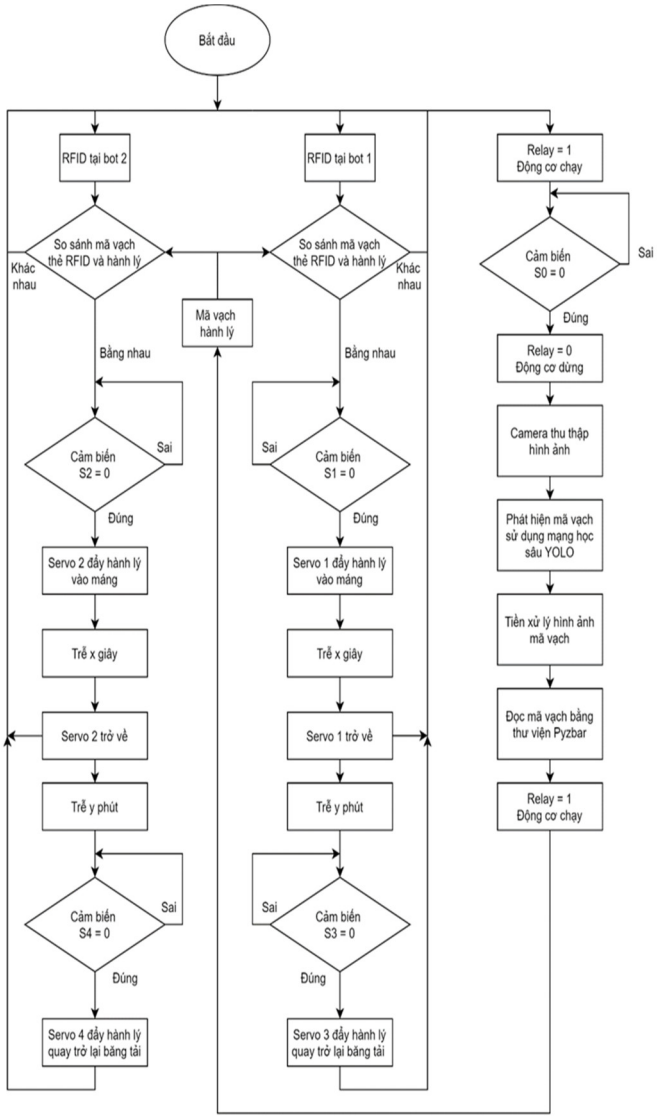
viện này Pyzbar mà chỉ sử dụng nó như công cụ trung gian khi đánh giá hiệu quả của toàn hệ thống thực tế. Kết quả thử nghiệm được trình bày chi tiết trong mục 3.2.

2.3. Xây dựng hệ thống tổng thể

Mô hình phần cứng của hệ thống băng tải giao hành lý thông minh hoàn thiện như minh họa trong hình 7.



Hình 7. Mô hình thực tế



Hình 8. Lưu đồ thuật toán tích hợp hệ thống giao hành lý thông minh

Mô hình hệ thống băng tải giao hành lý thông minh của chúng tôi được tích hợp với lưu đồ thuật toán như minh họa trong hình 8.

Việc đánh giá hiệu quả của toàn bộ hệ thống được thực hiện trên mô hình với 200 ảnh. Kết quả thử nghiệm được tính với hai tiêu chí là độ chính xác của các lần thử nghiệm thứ i (Acc_i) có tất cả 8 bước sau đều phải đúng gồm: (1) phát hiện có hành lý, (2) tác động dừng băng tải, (3) chụp ảnh mã vạch, (4) phát hiện mã vạch, (5) giải mã mã vạch, (6) so khớp mã vạch với mã RFID, (7) hành lý đẩy ra máng chứa, (8) đẩy hành lý trở lại băng tải nếu hành khách không lấy đồ sau một khoảng thời gian đặt trước. Độ chính xác được tính cho tất cả 200 lần như trong công thức sau:

$$Acc = \frac{\sum_{i=1}^{N=200} Acc_i}{N}$$

Ngoài ra, đối với mô hình thực tế chúng tôi còn tính độ chính xác từ bước (3) đến bước (5) với mỗi lần thực hiện có độ chính xác x_i và thời gian thực hiện t_i thì độ chính xác thử nghiệm N = 200 lần và thời gian trung bình được tính như trong các công thức sau:

$$x = \frac{\sum_{i=1}^{N=200} x_i}{N}$$

$$t_{tb} = \frac{\sum_{i=1}^{N=200} t_i}{N}$$

Kết quả đánh giá hệ thống sẽ được thực hiện và trình bày chi tiết trong mục 3.

3. KẾT QUẢ THỬ NGHIỆM

Tác giả thực hiện các thử nghiệm trên máy tính máy tính sử dụng CPU Intel Core i5-11400H, GPU NVIDIA GeForce GTX 1650, bộ nhớ Ram 8GB. Ngôn ngữ lập trình để thực hiện quá trình thu thập phát hiện, nhận dạng mã vạch và kết nối mạng là Python. Ngôn ngữ lập trình phần cứng cho Adruino là C++. Các thử nghiệm phát hiện mã vạch đánh giá trên hai pha: pha trực tuyến và pha không trực tuyến. Ở pha không trực tuyến nhóm tác giả sử dụng CSDL tự thu thập BarcodeEpu [1]. Ở pha trực tuyến tác giả sử dụng hình ảnh thu thập trực tiếp từ camera thông thường.

Các thử nghiệm được tiến hành trong nghiên cứu này gồm: (1) Khảo sát, đánh giá quá trình phát hiện mã vạch thông qua nhiều phiên bản YOLO khác nhau; (2) Đánh giá độ chính xác và thời gian đáp ứng toàn hệ thống;

3.1. Kết quả phát hiện mã vạch

Trong mục này, tác giả thực hiện chia dữ liệu BarcodeEPU [1] thành 10 phần bằng nhau. Sau đó, chúng tôi sử dụng phương pháp "Leave-one-subject-out" [12] để chia toàn bộ dữ liệu gồm 9000 ảnh mã vạch thành 10 phần hoàn toàn khác nhau, mỗi phần được xem như một đối tượng (subject) như trong phương pháp [12]. Sau đó, tại mỗi lần thử nghiệm, chúng tôi thực hiện huấn luyện mô hình (training model) với 8 phần dữ liệu thử nghiệm tương ứng với 7200 ảnh, kiểm chứng mô hình (validation model) với 900 ảnh tương ứng với một phần, và thử nghiệm mô hình (testing model) với 900 ảnh còn lại tương ứng với một phần. Điều đó có nghĩa là các dữ liệu huấn luyện, kiểm

chứng và thử nghiệm là không trùng nhau ở mỗi lần thử nghiệm. Ở lần huấn luyện tiếp theo, dữ liệu được bốc theo nguyên lý của giải pháp [12] cho đến khi đủ 10 lần. Kết quả mỗi lần thử nghiệm được lấy dựa trên độ chính xác của 900 ảnh thử nghiệm mô hình. Sau khi thực hiện đủ 10 lần thì kết quả cuối cùng của toàn bộ dữ liệu sẽ được tính bằng trung bình của 10 lần thử nghiệm trên. Trong đó, các mô hình thực hiện thử nghiệm với mẻ (batch size) là 64; tốc độ huấn luyện (learning rate) là 10^{-4}; số lần lặp để huấn luyện mô hình (epochs) là 300. Các thử nghiệm thực hiện sử dụng các phiên bản khác nhau từ YOLO V2 tới YOLO V7 bản tiny. Kết quả thử nghiệm gồm độ chính xác (Precision), độ triệu hồi (Recall), và độ chính xác trung bình (mAP) được trình bày như trong bảng 1.

Bảng 1. Bảng kết quả phát hiện mã vạch trên các phiên bản YOLO

Phương pháp	Precision (%)	Recall (%)	mAP (%)
YOLO V2	**96,0**	**95,0**	**94,1**
YOLO V3	98,1	96,9	98,3
YOLO V4	98,3	98,8	98,3
YOLO V5	**99,8**	**99,9**	**99,8**
YOLO V6	99,5	98,6	99,1
YOLO V7	97,6	98,2	98,6

Kết quả trong bảng 1 cho thấy, phiên bản YOLO V5 hiệu quả nhất đối với CSDL BarcodeEpu. Kết quả đạt được với độ chính xác, độ triệu hồi, và chỉ số mAP đạt lần lượt là 99,8%; 99,9%; và 99,8%. Phiên bản YOLO V2 đạt kết quả kém nhất trong tất cả các tiêu chí trên tại 96,0%; 95,0% và 94,1%. Một điều đặc biệt là không hẳn cứ các phiên bản cao hơn thì sẽ đạt kết quả cao hơn trong mọi bộ CSDL. Đối với BarcodeEpu của chúng tôi thì các phiên bản V6 và V7 lại có kết quả thấp hơn V5.

Ngoài việc so sánh kết quả độ chính xác như trên, trong nghiên cứu này tác giả còn tiến hành thực hiện khảo sát về thời gian thực hiện của các phiên bản YOLO để phát hiện một mã vạch. Đánh giá thử nghiệm các phiên bản YOLO chạy trên nền tảng CPU và GPU. Kết quả thời gian đáp ứng được trình bày trong bảng 2.

Bảng 2. Tốc độ phát hiện mã vạch trên các phiên bản YOLO

Phương pháp	Tốc độ xử lý một khung hình (CPU)	Tốc độ xử lý một khung hình (GPU)
YOLO V2	48,62ms	12,46ms
YOLO V3	75,59ms	14,40ms
YOLO V4	19,99ms	24,99ms
YOLO V5	48,30ms	14,61ms
YOLO V6	75,68ms	19,72ms
YOLO V7	95,16ms	23,28ms

Kết quả trong bảng 2 cho thấy mạng YOLO V5 có thời gian phát hiện khi sử dụng GPU là thấp nhất cùng nhóm với YOLO V3 (14ms) và khi sử dụng CPU thì kết quả cũng cách xa các kết quả cao nhất là 48ms so với 75ms và 95ms. Phiên bản YOLO V5 phù hợp với bài toán phát hiện mã vạch tác giả đề xuất. Do đó, trong phần thử nghiệm toàn bộ mô hình tiếp theo, tác giả sẽ sử dụng YOLO V5.

3.2. Đánh giá hệ thống trực tuyến

Trong phần này, tác giả sẽ tiến hành đánh giá trên hệ thống thực tế khi mô hình YOLO V5 [6] đã được huấn luyện trên toàn bộ CSDL BarcodeEpu [1]. Việc đánh giá được thực hiện trên 200 mã vạch dán sẵn trên các hộp như minh họa trên hình 7. Độ phân giải ảnh thực tế thu nhận 1080x1920 điểm ảnh. Tốc độ thu nhận ảnh 30 fps; YOLO chạy trên máy tính nền tảng sử dụng CPU Intel Core i5-11400H và GPU NVIDIA GeForce GTX 1650, bộ nhớ Ram 8GB. Kết quả đánh giá gồm: Độ chính xác phát hiện mã vạch kết hợp với độ chính xác giải mã mã vạch (x%); Thời gian trung bình từ khi thu nhận ảnh đến khi giải mã xong mã vạch (t_{tb}). Kết quả hai giá trị này được thể hiện như trong bảng 3.

Bảng 3. Kết quả đánh giá trực tuyến phát hiện và giải mã với YOLO V5[6]

CPU		GPU	
Độ chính xác - x(%) (phát hiện +giải mã)	Thời gian đáp ứng t_{tb}(ms)	Độ chính xác x-(%) (phát hiện +giải mã)	Thời gian đáp ứng t_{tb}(ms)
99,5	59,86	100	16,91

Kết quả này cho thấy hệ thống phát hiện với độ chính xác khá cao lên tới 100% khi sử dụng GPU. Thời gian đáp ứng khá nhanh, trung bình là 16,91ms, xấp xỉ 59fps cho một ảnh mã vạch. Ngoài ra, độ chính xác (Acc) chúng tôi thực hiện đạt 99,5% và tác giả nhận thấy những lần sai là do cơ cấu tác động phần cứng bị nhiễu và dừng băng tải sớm khiến cho ảnh chụp lệch không bắt được mã vạch. Do vậy, giải pháp đề xuất sử dụng YOLO và Pyzbar để phát hiện và giải mã mã vạch trong hệ thống thông minh là hoàn toàn phù hợp để triển khai ứng dụng trong thực tế.

Đối với hệ thống mô hình thử nghiệm tổng thể, tác giả thực hiện tính thời gian từ sau khi hệ thống phát hiện và giải mã mã vạch xong thì máy tính thực hiện gửi lệnh hoàn tất xuống mạch điều khiển trung tâm trung bình cho 200 lần thử nghiệm là 14,58ms. Do máy tính kết nối với mạch điều khiển trung tâm qua cáp với chuẩn truyền tin nối tiếp UART nên đạt độ chính xác cũng như thời gian đáp ứng khá nhanh. Sau đó, thời gian từ khi mạch điều khiển trung tâm thực hiện kết hợp kết quả đầu ra của mã vạch từ máy tính với bước giải mã RFID để gửi lệnh điều khiển cho các cánh tay gạt ở 200 lần thử nghiệm trung bình mất 8,6ms. Từ kết quả trên cho thấy, mỗi lần thực hiện toàn hệ thống trung bình mất khoảng thời gian là 83,31ms khi sử dụng CPU và 40,09ms khi sử dụng GPU. Thời gian đáp ứng toàn hệ thống khá nhanh, đảm bảo cả tính chính xác và tính thời gian thực. Điều này đã thể hiện tính khả thi khi triển khai thực tế của hệ thống đề xuất. Hệ thống đề xuất được tác giả xây dựng dưới dạng mô hình và cho chạy các chức năng như trong bài báo và để tại địa chỉ như trong [21].

4. KẾT LUẬN

Trong bài báo này, tác giả đã thiết kế một hệ thống điều khiển giao hành lý sân bay thông minh kết hợp giữa kỹ

thuật điện tử, công nghệ xử lý ảnh, kỹ thuật học sâu để phát hiện và giải mã mã vạch. Hệ thống của chúng tôi được cài đặt và thử nghiệm cả trực tuyến và không trực tuyến để kiểm tra độ chính xác cũng như đánh giá thời gian đáp ứng. Tuy nhiên, hệ thống chỉ được cài đặt và đánh giá trên mô hình mà chưa phải hệ thống thực tế. Trong thời gian tới, tác giả mong muốn ý tưởng thiết kế có cơ hội triển khai để đánh giá trên hệ thống thực với số lượng thử nghiệm nhiều hơn nữa. Nghiên cứu cũng thu thập, gán nhãn và công bố một bộ CSDL về mã vạch và chia sẻ hoàn toàn miễn phí cho cộng đồng nghiên cứu. Ngoài ra, nghiên cứu cũng có thể giúp cho sinh viên các ngành kỹ thuật, đặc biệt là sinh viên các ngành điều khiển và tự động hóa có thể vận dụng các kiến thức từ lý thuyết áp dụng vào thực thực tế. Các trải nghiệm thực tế này không chỉ giúp sinh viên hiểu kỹ hơn về một ứng dụng chuyên ngành mà còn giúp sinh viên biết cách triển khai, áp dụng các kiến thức mới của trí tuệ nhân tạo và xử lý ảnh để thông minh hóa các hệ thống điều khiển và tự động hóa hiện nay.

TÀI LIỆU THAM KHẢO

[1]. Huong-Giang Doan, 2022. *BarcodeEPU: https://zenodo.org/record/7455860.*

[2]. Redmon Joseph, Divvala Santosh, Girshick Ross, Farhadi Ali, 2016. *YOLOv1: You Only Look Once: Unified, Real-Time Object Detection*. 779-788. 10.1109/CVPR.2016.91.

[3]. Redmon Joseph, Farhadi Ali, 2017. *YOLO9000: Better, Faster, Stronger*. 6517-6525. 10.1109/CVPR.2017.690.

[4]. J. Redmon, A. Farhadi, 2018. *Yolov3: An incremental improvement*. CoRR journal, Vol. abs/1804.02767, pp. 1-6.

[5]. A. Bochkovskiy, C.Y. Wang, H.Y. M. Liao, 2004. *Yolov4: Optimal speed and accuracy of object detection*. ArXiv journal, Vol. abs/2004.10934, 2020, pp. 1-17.

[6]. Glenn Jocher, 2020. *Yolov5 in pytorch*. https://github.com/ultralytics/yolov5.

[7]. Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, Xiaolin Wei, 2022. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. Computer Vision and Pattern Recognition.

[8]. Wang Chien-Yao, Bochkovskiy Alexey, Liao Hong-yuan, 2022. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 10.48550/arXiv.2207.02696.

[9]. J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, L. FeiFei, 2009. *Imagenet: A large-scale hierarchical image database.* In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. IEEE, 2009

[10]. https://docs.arduino.cc/hardware/mega-2560

[11]. https://github.com/tzutalin/labelImg

[12]. Dang-Manh Truong, Huong-Giang Doan, Thanh-Hai Tran, Hai Vu, Thi-Lan Le, 2019. *Robustness Analysis of 3D Convolutional Neural Network for Human Hand Gesture Recognition*. International Journal of Machine Learning and Computing (IJMLC), Vol. 9, No. 2, pp.135-142.

[13]. https://pypi.org/project/pyzbar/

[14]. Pandi Thanapal, Prabhu John, Jakhar Mridula, 2017. *A survey on barcode RFID and NFC*. IOP Conference Series: Materials Science and Engineering, 263. 042049. 10.1088/1757-899X/263/4/042049.

[15]. D. E. King, 2009. *Dlib-ml: A machine learning toolkit*. J. Mach. Learn. Res, pp. 1755–1758.

[16]. Ren Shaoqing, He Kaiming, Girshick Ross, Sun, Jian, 2015. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39.10.1109/TPAMI.2016.2577031.

[17]. https://www.airportconveyors.eu/

[18]. Lin TY., et al., 2014. *Microsoft COCO: Common Objects in Context*. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham.

[19]. https://annefou.github.io/IoT_introduction/02-ESP8266/

[20]. Everingham M., Van Gool L., Williams C.K.I.. et al. *The PASCAL Visual Object Classes (VOC) Challeng*e. Int J Comput Vis 88, 303–338 (2010).

[21]. https://www.youtube.com/watch?v=0L15YKDuDQ0

AUTHOR INFORMATION

Doan Thi Huong Giang

Faculty of Control and Automation, Electric Power University